Heap Sort

Introduction

Heap sort divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element from it and inserting it into the sorted region. Heap sort maintains the unsorted region in a heap data structure to quickly find the largest element in each step [4].

Performance

In general, the performance of the sorting algorithm is analyzed from two aspects, time complexity and space complexity. Time complexity represents the time taken to execute the algorithm, which is generally considered in three cases: best case, worst case, and average case [6]. Space complexity represents the amount of memory required to complete a program [5].

Use **array** as data structure and **n** denotes the input array size, heap sort performance is as follows [4]:

Worst-case time complexity	$\boldsymbol{O(n\ log\ n)}$
Average time complexity	$\boldsymbol{O(n\ log\ n)}$
Best-case time complexity	$\boldsymbol{O(n\ log\ n)}$
Worst-case space complexity	$\boldsymbol{O(n)}$

Pseudocode [3]

Algorithm: HeapSort(Arr)

Input: an array of integers Arr

Output: The result of sorting Arr

```
for i = Arr.length - 1 to   do
maxHeapify(Arr, 0, i)
  swap(0, i)
end for
return Arr
```

Algorithm: maxHeapify(Arr, index, heapSize)

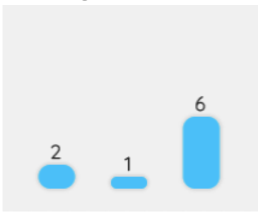
Input: an array of integers Arr, an integer index, an integer heapSize

Output: The result of sorting Arr

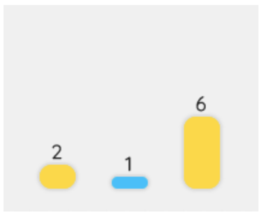
```
iMax = 0, iLeft = 0, iRight = 0
while (true) {
  iMax = index;
  iLeft = 2 * index + 1
  iRight = 2 * (index + 1)
  if iLeft < heapSize && Arr[index].value < Arr[iLeft].value then
     iMax = iLeft
  end if
  if iRight < heapSize && Arr[iMax].value < Arr[iRight].value then
     iMax = iRight
  end if
  if iMax != index then
     swap(Arr, iMax, index)
     index = iMax
   else
     break
   end if
end while
```

Example

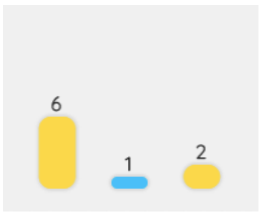
Sorting the list: 2,1,6



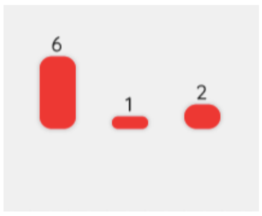
(1)



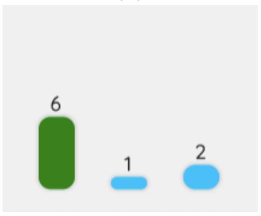
(2)



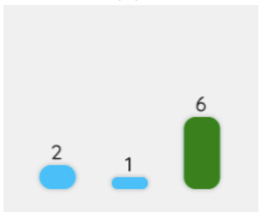
(3)



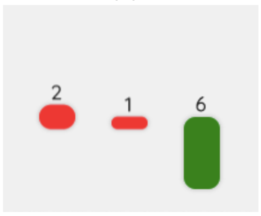
(4)



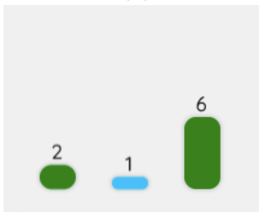
(5)



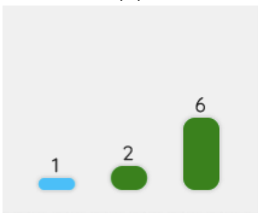
(6)



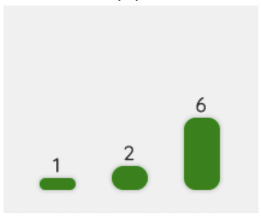
(7)



(8)



(9)



(10)

Implement in programming language

Java[1]

```java
public void heapSort(int arr[]) {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);
        for (int i=n-1; i>=0; i--)
        {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            heapify(arr, i, 0);
        }
    }

public void heapify(int arr[], int n, int i)
    {
        int largest = i;  // Initialize largest as root
        int l = 2*i + 1;  // left = 2*i + 1
        int r = 2*i + 2;  // right = 2*i + 2

        // If left child is larger than root
        if (l < n && arr[l] > arr[largest])
            largest = l;

        // If right child is larger than largest so far
        if (r < n && arr[r] > arr[largest])
            largest = r;

        // If largest is not root
        if (largest != i)
        {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            // Recursively heapify the affected sub-tree
            heapify(arr, n, largest);
        }
    }
```

JavaScript[2]

```
var len;
function buildMaxHeap(arr) {
    len = arr.length;
    for (var i = Math.floor(len/2); i >= 0; i--) {
        heapify(arr, i);
    }
}
function heapify(arr, i) {
    var left = 2 * i + 1,
        right = 2 * i + 2,
        largest = i;
    if (left < len && arr[left] > arr[largest]) {
        largest = left;
    }
    if (right < len && arr[right] > arr[largest]) {
        largest = right;
    }
    if (largest != i) {
        swap(arr, i, largest);
        heapify(arr, largest);
    }
}
function swap(arr, i, j) {
    var temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
function heapSort(arr) {
    buildMaxHeap(arr);
    for (var i = arr.length-1; i > 0; i--) {
        swap(arr, 0, i);
        len--;
        heapify(arr, 0);
    }
    return arr;
}
```

C[2]

```c
void swap(int *a, int *b) {
    int temp = *b;
    *b = *a;
    *a = temp;
}

void max_heapify(int arr[], int start, int end) {
    int dad = start;
    int son = dad * 2 + 1;
    while (son <= end) {
        if (son + 1 <= end && arr[son] < arr[son + 1])
            son++;
        if (arr[dad] > arr[son])
            return;
        else {
            swap(&arr[dad], &arr[son]);
            dad = son;
            son = dad * 2 + 1;
        }
    }
}

void heap_sort(int arr[], int len) {
    int i;
    for (i = len / 2 - 1; i >= 0; i--)
        max_heapify(arr, i, len - 1);
    for (i = len - 1; i > 0; i--) {
        swap(&arr[0], &arr[i]);
        max_heapify(arr, 0, i - 1);
    }
}
```

References

1. GeeksforGeeks Contributors (2018). *Java Program for Heap Sort.* [Online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/java-program-for-heap-sort/ [Accessed 6 Mar. 2021].
2. Runoob Contributors (2018). *Heap sort*. [Online] Runoob. Available at: https://www.runoob.com/w3cnote/heap-sort.html [Accessed 6 Mar. 2021].
3. Visualgo Contributors (2014). [Software] Visualgo. Available at: https://visualgo.net/en/sorting [Accessed 6 Mar. 2021].
4. Wikipedia Contributors (2021). *Heapsort*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Heap_sort [Accessed 6 Mar. 2021].
5. Wikipedia Contributors (2021). *Space complexity*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Space_complexity [Accessed 20 Mar. 2021].
6. Wikipedia Contributors (2021). *Time complexity*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Time_complexity [Accessed 20 Mar. 2021].